mathematics

MDPI

Article

New Algorithm to Solve Mixed Integer Quadratically Constrained Quadratic Programming Problems Using Piecewise Linear Approximation

Loay Alkhalifa [1,*] and Hans Mittelmann [2]

[1] Department of Mathematics, College of Sciences and Arts, Qassim University, Ar Rass 51921, Saudi Arabia
[2] School of Math & Stat Sciences, Arizona State University, Tempe, AZ 85287, USA; mittelmann@asu.edu
* Correspondence: loay.alkhalifa@qu.edu.sa

Abstract: Techniques and methods of linear optimization underwent a significant improvement in the 20th century which led to the development of reliable mixed integer linear programming (MILP) solvers. It would be useful if these solvers could handle mixed integer nonlinear programming (MINLP) problems. Piecewise linear approximation (PLA) is one of most popular methods used to transform nonlinear problems into linear ones. This paper will introduce PLA with brief a background and literature review, followed by describing our contribution before presenting the results of computational experiments and our findings. The goals of this paper are (a) improving PLA models by using nonuniform domain partitioning, and (b) proposing an idea of applying PLA partially on MINLP problems, making them easier to handle. The computational experiments were done using quadratically constrained quadratic programming (QCQP) and MIQCQP and they showed that problems under PLA with nonuniform partition resulted in more accurate solutions and required less time compared to PLA with uniform partition.

Keywords: mixed integer nonlinear programming; piecewise linear approximation; branch and bound

Citation: Alkhalifa, L.; Mittelmann, H. New Algorithm to Solve Mixed Integer Quadratically Constrained Quadratic Programming Problems Using Piecewise Linear Approximation. *Mathematics* **2022**, *10*, 198. https://doi.org/10.3390/math10020198

Academic Editors: Humberto Rocha and Ana Maria Rocha

Received: 23 November 2021
Accepted: 4 January 2022
Published: 9 January 2022

1. Introduction

The rapid advances of Linear Programming (LP), Non-Linear Programming (NLP), and Mixed Integer Linear Programming (MILP) techniques and algorithms in the 20th century led to the development of robust MILP solvers that can easily handle problems with millions of variables. On the other hand, methods that deal with Mixed Integer Non-Linear Programming (MINLP), which is the most difficult class of optimization, started to improve recently. Even with the current improvements in the MINLP solvers, it would be a great step forward if MINLP problems could be solved globally by MILP solvers. In principle, this can be done by approximating the MINLP problem by an MILP one, but solving this approximation by an MILP solver will probably be harder than solving the original problem by an MINLP solver. Recent discussions about software solvers and their development can be found in [1,2]. More detail about MINLP and their algorithms can be found in [3–6], and a recent general survey was introduced by [7]. For some real life optimization applications, we refer to [8–11].

One of the methods to remodel MINLP as MILP is the piecewise linear approximation (PLA) ([7]). This method takes an advantage of the fact that any continuous function can be approximated by a piecewise linear one. Replacing every nonlinear function in the MINLP model by their PLAs will yield an MILP model. Because of the size of the new approximated model, the PLA was not introduced to do full approximations to the MINLP models. Instead, it was used mostly to find linear under/over estimators to some of the functions involved in the models.

The procedures of approximating a nonlinear function $f(x)$, where $x \in [x_l, x_u] \subseteq \mathbb{R}$, by a piecewise Linear (PL) function are simple. First, the domain of the variables x is

divided into n intervals by introducing the breakpoints $x_0 = x_l < x_1 < x_2 < \cdots < x_n = x_u$. Then the function f is evaluated at each breakpoint, and the lines that connect the points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$ form the desired PL function, denoted by $\overline{f}$. The PLA idea was also extended to higher dimensional functions.

Increasing the number of breakpoints will increase the accuracy of the approximation, but it will result in problem size growth. This major drawback may restrict the PLA benefits to functions of only few dimensions. Ref. [12] discuss minimizing the number of breakpoints needed to approximate a nonlinear function up to a given tolerance. This subject will not be discussed here. When doing PLA, introducing breakpoints requires adding both binary and continuous variables to the optimization problem in addition to new constraints. This is done by setting the binary variables to be SOS1 or SOS2 (please see [13]). The number of the new variables and constraints may increase exponentially with the number of dimensions of the function and it varies depending on the model used to do the PLA.

The following section will present a brief literature review on the PLA modeles. The remainder of the paper is organized as follows: in Section 2, few approaches on how to improve the PLA models are introduced. One approach shows how to take an advantage of a local solution to choose the breakpoints by nonuniform partitioning, and our contribution will be using this partitioning to produce better PLA. The other approach is to apply the PLA on only a part of the optimization problem. The computational results are given in Section 3, where the tests are applied to continuous and discrete problems.

Literature Review

PLA dates back to the 1950s (see, for example, [14–16]), and since then, many PLA models were introduced. The convex combination model (CC) (also called the λ-model in some sources) is one of the most common PLA models and it was introduced by [15]. Another model that was introduced by [14] is the incremental model. The incremental model requires one less binary variable and one less continuous variable compared to the CC model, but it needs nearly twice the number of new constraints. Modifications were done to the CC model and the incremental model resulting in the disaggregated convex combination model ([16,17]) and the multiple choice model ([18]), respectively. Piecewise linear relaxation techniques are widely used by many algorithms, and these techniques are different from PLA, even though they share some steps. In this paper, we are interested only on PLA, and we refer to [19–21], for further reading about piecewise linear relaxations.

The efficiency of MILP solvers will be affected if the PLA is done using many breakpoints, especially for higher dimensional functions. More accurate PLA can be obtained by increasing the number of breakpoints, but in all models mentioned above, the introduced binary variables will be almost as many as the breakpoints. Therefore the MILP solvers might not be able to deal with the size of the resulting MILP problem. Introducing less breakpoints will result in small approximated problems but it might lead to a bad approximation.

Many attempts have been made to deal with the size issues, but these attempts do not resolve the problem completely. A major improvement in this area took a place when [22,23] introduced a technique that allows PLA models to use dramatically fewer binary variables. They applied the technique to both versions of the convex combination model and denoted it by logarithmic model, and it was later applied to other PLA models. To do the PLA with $n+1$ breakpoints using the logarithmic model, only $\lceil \log_2 n \rceil$ binary variables are required, instead of approximately n required by other PLA models. In this paper, we will use the CC and logarithmic disaggregated convex combination (LOG) models to do the computational experiments on quadratically constrained quadratic programming (QCQP) and Mixed Integer QCQP (MIQCQP).

A good overview of the models mentioned in this section is presented by [24], and it was shown that they are equivalent in terms of the feasibility of their solutions. In term of tightness, it is desired for a PLA model to be locally ideal (the vertices of its LP relaxation satisfy the integrality constraints of the original MINLP problem). Ref. [25] shows that

the incremental model is superior to the CC model since the incremental model is locally ideal (this is supported by the computational comparison that will be discussed below). Ref. [23] proves that all PLA models mentioned earlier are locally ideal except the CC model. However, the CC model has the sharpness property, i.e,. the projection of the vertices of the model onto the original set of the variables is exactly the convex hull of the set. Moreover, it can be shown that any locally ideal model is sharp. More recent theoretical comparison between the models was given by [26].

In the computational experiments made by [23], they approximated functions with one variable in 100 test instances, and used CPLEX to solve the PLAs. It was observed that for less than 10 breakpoints, all models performed well with the multiple choice model being slightly better. As the number of breakpoints increases, the logarithmic models started to gain the upper hand. When 33 breakpoints were used, the logarithmic models are almost 20 times faster than the incremental model, which is more than two times faster than the CC and multiple choice models. The disaggregated convex combination is much slower than the rest. Similar outcomes resulted from testing 100 problems where the approximated functions have two variables. A less extensive experiment was done by [27], and it was concluded that in some cases, it is better to use the incremental model rather than the logarithmic one, even though the latter has smaller size.

2. Improving PLA Models

In this section, a few approaches to improve the PLA models will be presented. Since PLA requires introducing many binary and continuous variables and constraints, it was not studied as a stand-alone method to solve MINLP problems until recently. It is usually used as a tool in optimization algorithms to find local solutions or to under/over estimate some of the nonlinear functions. For PLA models to completely transform an already hard MINLP problem and produce an MILP problem that is easier than the original to be handled, more improvements on these models are needed. Note that even if the targeted problems in this paper have many variables, the PLA models will be applied separately only to functions of two variables or less.

One approach to improve the models is to choose the breakpoints such that the variable domain is nonuniformly partitioned. This approach was motivated by the desire to produce an accurate approximation with reasonable problem size. Unfortunately, this is rarely possible since an accurate approximation requires many breakpoints. One of the methods to overcome this issue is to study how to partition a domain. Most existing PLA models choose the locations of breakpoints based on a uniform partitioning of the domain. The approach proposed in this section, namely a nonuniform partitioning, leads to a better PLA model than the one produced by uniform partitioning with the same number of breakpoints. The details are given in Section 2.1.

Another approach that will be presented in this section is applying the PLA partially to problems with many nonlinear functions. Given a complicated MINLP problem, applying the PLA to only some of its nonlinear constraints will not make the problem solvable by an MILP solver, but it might make the problem less complicated for MINLP solvers. The idea is to identify complicated nonlinear constraints and approximate some of them by linear ones, then the modified problem is solved using MINLP solvers. An algorithm that identifies these constraints and approximates them will be provided in Section 2.2.

2.1. Choosing Breakpoints by Nonuniform Partitioning

Most PLA models, that are used within the context of optimization, rely on uniform partitioning of variable domains. A few methods to do nonuniform partitioning were introduced within other contexts. For example, [28] used nonuniform partitioning to approximate a one dimensional curve. The idea is to add more breakpoints in the part of the domain where the function has higher curvature. This method is done by solving a shortest path problem, and it was later used within the PLA approach for one and two

dimension by [29]. Other methods were introduced to use nonuniform partitioning to get piecewise convex/linear relaxations, such as the one proposed by [30].

The partitioning method suggested in this section is to build the partitioning around a local solution. The density of breakpoints increases as they get closer from both sides to the local solution, which itself is a breakpoint. Assume the local solution of a variable $l \leq x \leq u$ is x^*, then a possible partitioning is to have a breakpoint in the halfway in the interval from the upper/lower bound to x^*, then another point halfway toward x^* and so on. This partitioning can be given by using the formulas

$$x^* + \frac{u - x^*}{k^i} \qquad \text{and} \qquad x^* - \frac{x^* - l}{k^i}, \tag{1}$$

for $k = 2$ and $i = 1, 2, \ldots$, to get the breakpoint values on both sides of x^*, as shown in Figure 1A.

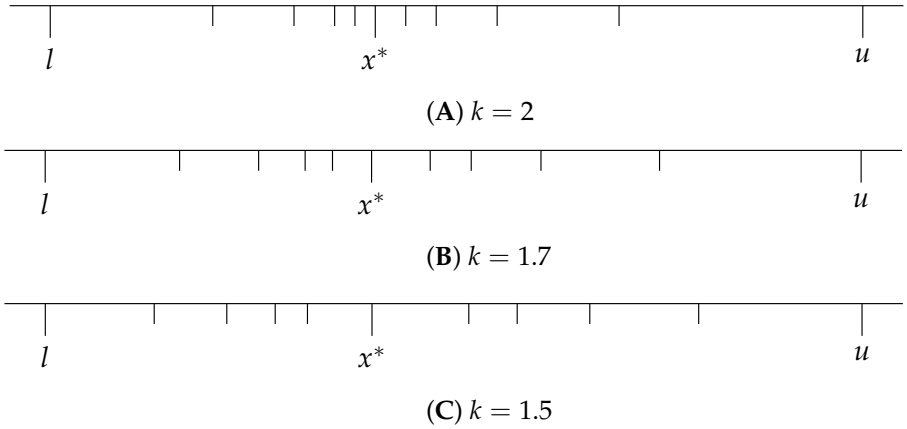

Figure 1. Partitioning Using the Formulas $x^* + \frac{u-x^*}{k^i}$ and $x^* - \frac{x^*-l}{k^i}$.

The problem with the case $k = 2$ is that it leaves half of the interval between the upper/lower bound and x^* without any breakpoints, which may affect the accuracy of the PLA. Thus, giving k different values between one and two, as in Figure 1B,C, may yield better partitioning. It can be noticed that as k gets closer to one, the density of breakpoints shifts away from x^*. If x^* happened to be at or very close to one of the bounds, the partitioning will be on one side only. In the case that the approximated function has two variables, the same logic is applied to both domains.

The PLA using this partitioning was tested for many values of $1 < k < 2$ against PLA with uniform partitioning. The results show that models with nonuniform partitioning were solved faster and yielded better solutions to most of the tested instances. Details about the targeted optimization problems and the test results will be given in Section 3.

2.2. Partial PLA

In this section, PLA will be performed only on parts of a given MINLP problem, instead of approximating all nonlinear functions. This is done by targeting complicated nonlinear functions be handled by PLA, leaving the remaining functions unchanged. The goal of this approach is to avoid introducing unnecessary variables that result from approximating simple functions. To test this approach, an algorithm is introduced to deal with problems having many nonlinear constraints by picking one constraint at every iteration and approximating it, until enough constraints are replaced.

Assume the algorithm is applied to an MINLP problem with x^* and f^* as its global optimal solution and objective function value, respectively. The algorithm starts by solving the problem (before doing any PLA) using an MINLP solver for a few nodes before it stops the solving process when a specified number of nodes, N, is reached. Then all nonlinear constraints are identified, and since the solving process was interrupted, using the current

node solution will probably result in some of these constraints being violated. Now all nonviolated constraint are considered to be easy since the solver was able to satisfy them within few nodes, so the PLA will not be applied to these constraints.

Now the algorithm picks a constraint from the violated ones to be approximated using a PLA model, and this constraint is suggested to be the most violated one. As a result, the problem now has one less nonlinear constraint. Then the process is repeated until the problem has no violated constraints. Note that some of the nonlinear constraint will not be replaced, so the problem is still an MINLP one. At this point, the problem is solved regularly to produce a solution $\overline{x}^*$ to the modified problem with a function value $\overline{f}^*$. $|f^* - \overline{f^*}|$ is evaluated and the solving times of the original and modified problems are compared. Algorithm 1 shows the steps of the partial PLA approach on MIQCP problems, for some $\epsilon > 0$ and $k \in \mathbb{N}$.

Algorithm 1: An algorithm that chooses some constraints to be approximated.

```
Input:MIQCP problem;
Output:MIQCP problem with up to k constraints being approximated;
while |f̄* − f*| ≥ ε and iteration ≤ k do
    start the solving process;
    if solving is done before number of nodes reaches N then
        set the current objective value = f̄*;
        if |f̄* − f*| ≥ ε then
            add more breakpoints;
        end
    else
        stop the solving process when the tree has N nodes;
        identify the quadratic constraints;
        if number of violated constraints ≥ 1 then
            replace the most violated one with its PLA;
            set f̄* = ∞;
        else
            solve the PL problem and set the current objective value = f̄*;
        end
    end
end
```

With each iteration, one constraint is replaced by its PLA. Obviously, if k is large enough, then all violated constraints will be replaced and the resulting problem will probably be harder to solve because of the size. The computational experiments with large k came as expected and produced extremely large problems. Then k was set to one, i.e., PLA was applied to only the most violated constraint in the first iteration. For some test instances, both PLA models, that were introduced in the previous section, produced modified problems that were solved faster than the original ones, with similar solution and objective value. As the number of replaced constraints increases, the modified problem gets more complicated. Most of the tested instances appeared to be better off without PLA if $k \geq 3$. In Section 3, a summary of the test instances will be presented in addition to the main findings on applying PLA to only parts of these instances.

3. Computational Experiments

This section presents detailed reports on computational experiments that target QCQP and MIQCQP problems. The approaches that were introduced in the previous section can be applied to MINLP problems, but they were implemented to test the quadratic problems only. The same procedures can be applied to the MINLP problems by modifying the code that implements the procedures to allow it to identify other non linear functions beside the quadratic ones. Most of the chosen problems are not trivial and are based on

different real world applications. The QCQP problems were taken from the kall instances in the MINLPLib library (http://www.minlplib.org/index.html, accessed on 15 May 2021), whereas the MIQCQP problems were taken from both MINLPLib and QPLIB (http://qplib.zib.de/accessed on 15 May 2021). Two dimensional functions are involved in all of the instances, so 2d PLA is required.

The nonlinear functions in the instances are approximated by linear ones using the CC and LOG models. If the breakpoints needed for CC and LOG are determined using nonuniform partitioning, then the models will be called NCC and NLOG. These four models are also used to do partial PLA to the test instances by approximating all the nonlinear functions in one constraint.

The code that implements the PLA models were written using PySCIPOpt (http://scip-interfaces.github.io/PySCIPOpt/docs/html/index.html, accessed on 15 May 2021), which is a Python interface for the global solver SCIP. The tests were applied to the instances several times before taking the average results. After the instance is read by SCIP, all quadratic constraints (or general nonlinear constraints, in case the instance is an MINLP one) are identified before the domain of every variable involved in a quadratic function is partitioned (uniformly or nonuniformly). Then SCIP adds all necessary variables and constraints needed to replace the targeted function. Finally, any quadratic constraint is deleted from the instance and linear constraints, that approximate the deleted ones, are added. Now the instance is written in a format that can be read by MILP solvers.

The local instance solutions, that are needed for the nonuniform partitioning, are generated using the solver Knitro. The comparison between the uniform and nonuniform partitioning is determined through solving the MILP problems resulting from applying CC, LOG, NCC, and NLOG to the test instances. The MILP problems are solved using CPLEX 12.10 with its default settings, and the computations are done in a Linux machine with Intel Xeon E5-2620 2GHz processor. The time limit for each problem is set to one hour.

The partial PLA tests are done by solving the original instances and the same instances with one constraint replaced. The instances are solved using SCIP Optimization Suite 6.0.0 on a Linux machine with Intel Core i5-7y54 1.2GHz processor. The default solver settings were used with a time limit of two hours. The results of the computational experiments will be described in detail in the following sections.

3.1. Continuous Variables

The computational results that are reported in this section are for the kall instances, where the objective is linear and the constraints are linear and quadratic. The quadratic constraints contain up to two functions with two variables of the forms $(x + y)^2$ or xy. Moreover, one variable functions of the form x^2 appear in the constraints of some of the problems.

Statistics of the instances are given in Table 1, where seq# is the sequential number that will refer to the corresponding instance throughout the section. The number of variables is shown under #v, and #cons (q) represents the total number of constraints including (q) quadratic ones. The best objective values found so far (according to the MINLPLib library) are entered under the obj val column header. All objective values shown in the table are proven to be the global optimum by at least 3 global solvers except instances 2, 3, 5, and 12.

The instances were solved regularly by SCIP, with time limit of 2 h. The solving times are recorded and presented in seconds in the last column of Table 1. For some instances, the limits, machine memory (ML) or time (TL), were reached before the instance is solved to optimality. Moreover, it can be observed from the solving times that some instances are trivial and solved quickly but others took time to be solved. Solving the instances here was not intended to find their global solutions, since the best solutions are already found and listed in the library and no further improvement to the solutions can be done. Instead, they were solved to give an idea about their difficulty levels, and to have a reference when it is needed to compare their results to the modified problems results.

Table 1. Statistics of continuous test instances, and the solving time by SCIP.

Instance	seq#	#v	#cons (q)	obj val	Time
kall_circles_c6a	1	18	54 (22)	2.1117	1927
kall_circles_c6b	2	18	54 (22)	1.9736	2964
kall_circles_c7a	3	20	69 (29)	2.6628	2612
kall_circles_c8a	4	22	86 (37)	2.5409	TL
kall_congruentcircles_c31	5	10	16 (4)	0.6438	1
kall_congruentcircles_c32	6	10	16 (4)	1.3759	1
kall_congruentcircles_c41	7	12	24 (6)	0.8584	1
kall_congruentcircles_c42	8	12	24 (7)	0.8584	1
kall_congruentcircles_c51	9	14	34 (11)	1.073	12
kall_congruentcircles_c52	10	14	34 (11)	1.5371	4
kall_congruentcircles_c62	11	16	46 (16)	1.2876	16
kall_congruentcircles_c63	12	16	46 (16)	1.2876	11
kall_congruentcircles_c72	13	18	60 (22)	1.9663	225
kall_diffcircles_10	14	24	71 (45)	11.9355	6356 (ML)
kall_diffcircles_5a	15	14	24 (11)	5.1162	63
kall_diffcircles_5b	16	14	24 (11)	5.1162	44
kall_diffcircles_6	17	16	31 (16)	7.7879	102
kall_diffcircles_7	18	18	40 (22)	7.1531	177
kall_diffcircles_8	19	20	49 (28)	14.4813	3350
kall_diffcircles_9	20	22	60 (36)	13.3503	5118 (ML)

Before discussing the results of instance PLAs, one important factor about the instances has to be taken into account: the sizes of variable domains. When the variable involved in PLA of a function has a large domain, the domain will need more breakpoints to get a good approximation. However, introducing many breakpoints will affect the size, so the number of breakpoints for the computational tests is set to 10 for all domains. The domain size for each variable in the tested instances ranges between 1 and 18, with average size of 7.25 per domain. Therefore, 10 points should be enough to get good models for test purposes.

To compare between the PLA models with respect to the sizes of the produced problems, the instances were transformed into MILP problems by the models CC and LOG (using nonuniform partitioning would give the same size). Each instance was approximated by these models using 10, 20, and 30 breakpoints. Then the average numbers of constraints and binary/continuous variables per problem is recorded for each set of breakpoints, as shown in Table 2. It is apparent from the table that there is a significant advantage for the model LOG in terms of needed number of binary variables and constraints. Moreover, it will be shown later that this advantage is not outweighed by the CC model's advantage of having many fewer continuous variables.

Table 2. Average sizes per problem produced by CC and LOG models using 10, 20, and 30 breakpoints.

		10	20	30
BV	CC	6135	27,195	63,355
	LOG	303	377	414
CV	CC	3808	15,083	33,916
	LOG	22,738	90,416	203,417
Cons	CC	3983	15,259	34,092
	LOG	764	891	983

While Table 2 gives enough insight on the problem sizes, detailed statistics for every problem approximated using 10 breakpoints is given in Table 3. This table makes it easier to track the detail of every tested problem and compare between its size and computational

results. It should be expected that most problems will not be solved easily by CPLEX due to the large number of binaries and constraints.

Table 3. The sizes of the instances produced by CC and LOG models with 10 breakpoints.

p#	CC			LOG		
	BV	**CV**	**Cons**	**BV**	**CV**	**Cons**
1	6966	4318	4526	344	25,818	871
2	6966	4318	4526	344	25,818	871
3	9234	5720	5991	456	34,220	1152
4	11,826	7322	7678	584	43,822	1473
5	1134	710	744	56	4210	149
6	1134	710	744	56	4210	149
7	2106	1312	1376	104	7812	271
8	2106	1312	1376	104	7812	271
9	3402	2114	2218	168	12,614	433
10	3402	2114	2218	168	12,614	433
11	5022	3116	3270	248	18,616	635
12	5022	3166	3270	248	18,616	635
13	6966	4318	4532	344	25,818	877
14	14,742	9124	9535	728	54,624	1800
15	3402	2114	2208	168	12,614	423
16	3402	2114	2208	168	12,614	423
17	5022	3118	3255	248	18,616	620
18	6966	4318	4512	344	25,818	857
19	9234	5720	5977	456	34,220	1132
20	11,826	7322	7652	584	43,822	1447

3.1.1. Uniform vs. Nonuniform Partitioning

The computational experiments on solving the MILP problems approximating the selected instances were made by CPLEX with a time limit of one hour per problem. The objective of these experiments is to compare between uniform and nonuniform partitioning. The nonuniform partitioning is done using different values for the parameter k in the formulas in Equation (1).

Table 4 compares the results obtained from solving the CC and NCC problems, with $k = 1.5, 1.7$. For each model, the solving time is listed in seconds or as TL if the time limit is reached. The objective value of the best integer solution found within the time limit is listed under BI, and F means the solver failed to find an integer solution. Initially, 10 breakpoints were used by the models, but CPLEX failed to find a solution for most of the problems, so they were regenerated using 7 points. The third column measures the difference, if applicable, between the best integer value, $\overline{f^*}$ and the best value of the original instance, f^*, which can be found in Table 1. Smaller $|f^* - \overline{f^*}|$ value indicates that the approximation is acceptable. Note that for problems that reached time limit, the best integer value could keep improving if there is no time limit.

Given the fact that most of the original instances were solved by SCIP to global optimality within less than an hour (Table 1), it can be confirmed that complete transformation of MINLP problems into MILP ones will not make the problems easier, especially when approximating many functions. Nonetheless, this is not the goal of the experiments since the main purpose is to compare partitioning methods for PLA models. The aspects of the comparison between uniform and nonuniform partitioning will be the accuracy of the approximation and the time needed to solve the approximation.

Table 4. Results of solving MILP problems produced by CC and NCC models using 7 breakpoints.

p#	CC			NCC ($k = 1.5$)			NCC ($k = 1.7$)		
	Time	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$
1	TL	F	NA	TL	1.184	0.9277	TL	2.47	0.3583
2	TL	F	NA	TL	3.324	1.3504	TL	F	NA
3	TL	F	NA	TL	2.0317	0.6311	TL	2.6628	0
4	TL	F	NA	TL	F	NA	TL	F	NA
5	18	0.5724	0.0714	29	0.6009	0.0429	81	0.5847	0.0591
6	11	1.366	0.0099	8	1.3831	0.0072	5	1.3544	0.0215
7	1	0.8084	0.05	1	0.8584	0	1	0.8584	0
8	TL	0.8084	0.05	133	0.8269	0.0315	110	0.8052	0.0532
9	TL	1.5373	0.4643	TL	0.7367	0.3363	TL	0.5087	0.5643
10	1200	1.473	0.0641	TL	1.5371	0	TL	1.588	0.0509
11	TL	1.221	0.0666	TL	1.2876	0	TL	1.2876	0
12	856	1.118	0.1696	TL	1.2876	0	TL	1.2876	0
13	TL	1.6689	0.2974	TL	1.9663	0	TL	1.817	0.1493
14	TL	F	NA	TL	F	NA	TL	F	NA
15	TL	5.2619	0.1457	TL	5.1162	0	TL	6.479	1.3628
16	1400	2.838	2.2782	527	3.172	1.9442	TL	3.4476	1.6686
17	3400	7.5885	0.1994	TL	7.2298	0.558	TL	F	NA
18	TL	F	NA	TL	F	NA	TL	F	NA
19	TL	F	NA	TL	15.431	0.9497	TL	17.611	3.1297
20	TL	F	NA	TL	F	NA	TL	F	NA

The data listed under the time columns indicate that all models are close in terms of the number of problems that were solved within the time limit. However, for problems that reached the time limit, the model with uniform partitioning failed to find an integer solution in eight problems, compared to four problems for the NCC model with $k = 1.5$. This implies that problems generated by CC models with nonuniform partitioning usually find feasible solutions faster than with uniform partitioning. The same outcome resulted when other nonuniform cases were tested for $k = 1.4, 1.6, 1.8$. Even when both uniform and nonuniform cases resulted in a problem that produced an integer solution within the time limit, the gap in the nonuniform case is smaller most of the times. Therefore, it can be concluded that problems produced by NCC models are usually solved faster than the ones produced by CC models.

The quality of the approximation is better tested without the time limit, where the solver runs until the global solution is found, and then the solutions of the original and approximated problems are compared. In spite of that, the table provides enough data to compare the quality of CC and NCC approximations. It can be observed that the integer solution is the same as the solution of the original instance in many problem for both cases of nonuniform partitioning, while the uniform partitioning never produced an identical integer solution to the original one. Moreover, for the cases that the integer solution is not the same as the original one, the difference between the two solutions is mostly less in the NCC cases.

The computational results for solving problems produced by the models LOG and NLOG are summarized in Table 5. The test setting and comparison aspects for these models are the same as the CC and NCC models, except for the number of breakpoints where 11 is used for this test. The outcomes also turned out to be similar: the nonuniform partitioning improved the models in both speed and accuracy. Both NLOG models resulted in more problems that were solved within time limit, compared to LOG models; and when a problem is solved within time limit for all models, almost always the NLOG problem needs less solving time. Moreover, the difference $|f^* - \overline{f^*}|$ is mostly smaller in the NLOG cases. It should be mentioned that in nonuniform partitioning, having a local solution as one of the breakpoints helped improving the model's quality.

Table 5. Results of solving MILP problems produced with LOG and NLOG models using 11 breakpoints.

p#	LOG			NLOG ($k = 1.5$)			NLOG ($k = 1.7$)		
	Time	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$
1	TL	F	NA	TL	F	NA	TL	F	NA
2	TL	F	NA	TL	1.9488	0.0248	TL	F	NA
3	TL	F	NA	TL	2.6628	0	TL	2.6628	0
4	TL	F	NA	TL	F	NA	TL	F	NA
5	8	0.5435	0.1003	6	0.6009	0.0429	73	0.5847	0.0591
6	10	1.368	0.0079	2	1.3831	0.0072	157	1.3544	0.0215
7	3	0.8324	0.026	1	0.8584	0	47	0.8584	0
8	37	0.8348	0.0236	16	0.8269	0.0315	133	0.8052	0.0532
9	TL	2.944	1.871	TL	1.036	0.037	TL	0.378	0.695
10	1764	1.51	0.0271	194	1.5371	0	915	1.5371	0
11	TL	1.651	0.3634	1042	1.2876	0	821	1.2876	0
12	346	1.038	0.2496	231	1.2876	0	478	1.2876	0
13	TL	2.097	0.1307	TL	1.9663	0	TL	1.9663	0
14	TL	F	NA	TL	F	NA	TL	F	NA
15	2980	5.038	0.0782	TL	5.1162	0	161	4.6298	0.4864
16	TL	4.201	0.9152	191	3.172	1.9442	1625	3.4476	1.6686
17	TL	7.642	0.1459	450	7.2298	0.5581	2516	7.572	0.2159
18	TL	F	NA	TL	6.411	0.7421	TL	F	NA
19	TL	F	NA	935	7.182	7.2993	TL	2.288	12.1933
20	TL	F	NA	TL	7.217	6.1333	TL	F	NA

Comparing between the data in Tables 4 and 5 makes it clear that the logarithmic models are significantly better than the non logarithmic ones. This confirms the fact that for a PLA model, increasing the number of binary variables has more negative impact than increasing the continuous variables. Even though the logarithmic models used 4 points more (11 compared to 7), which considerably affects the size, the solving times turned out to be much less than the times for the non logarithmic case. Therefore, using logarithmic models allows introducing more breakpoints and, consequently, producing more accurate approximation. For example, the average difference between optimal value of the original instance and approximation, $|f^* - \overline{f^*}|$, per LOG problem is 0.328, while it is 0.667 for CC problems.

The computational experiments in this section have shown that full PLA of MINLP problems will not make the new problems any better, yet they prove that the improvement of PLA models is promising. In addition to the major development of the PLA model caused by logarithmic formulation, different components of the model can also be improved. For example, it was demonstrated that nonuniform partitioning based on local solutions can add more improvement to the models. Finally, the uses of PLA are not limited to the full PLA; it can be useful if applied to only parts of an MINLP problem, as will be shown in the next section.

3.1.2. Partial PLA

PLA was applied partially to different QCQP and MIQCQP using Algorithm 1. If the number of approximated constraints is large, then the modified problem gets more complicated and the original problem is better off without this approximation. However, in this section, the algorithm is applied with one iteration to many instances belonging to different groups from the MINLPLib library. Interestingly, this approach worked on some instances, and produced problems that needed shorter solving times. Table 6 summarizes the statistics of some of these instances, in addition to computational results of solving them using SCIP with 2 h time limit.

Table 6. Statistics of different test instances, and the solving results by SCIP.

Instance	seq#	#v	#cons (q)	obj val	Time	Gap %
kall_circlespolygons_c1p12	1	43	48 (21)	0.3396	TL	∞
kall_circlespolygons_c1p13	2	43	48 (21)	0.3396	TL	∞
kall_circlesrectangles_c1r12	3	49	41 (23)	0.3396	TL	∞
kall_diffcircles_5a	4	14	24 (11)	5.1162	63	0
kall_diffcircles_6	5	16	31 (16)	7.7879	102	0
pooling_foulds3stp	6	832	1089 (1024)	−8	5730	0
pooling_foulds4stp	7	832	1089 (1024)	−8	6235	0

As can be seen in the table, SCIP failed to close the gap when solving instances 1, 2, and 3; but it succeeded for the other problems. The goal of testing these instances is to show that approximating one constraint per instance aids the solver to handle the instance better. For each modified instance Table 7 presents the solving time, the optimal objective value $\overline{f^*}$, and the difference between the optimal values of the original and modified instance.

Table 7. Computational Results of Problems Produced by Partial PLA Using 10 Breakpoints.

p#	CC			LOG		
	Time	$\overline{f^*}$	$\lvert f^* - \overline{f^*}\rvert$	Time	$\overline{f^*}$	$\lvert f^* - \overline{f^*}\rvert$
1	92	0.2949	0.0447	241	0.2848	0.0548
2	258	0.2914	0.0482	358	0.3227	0.0169
3	TL	0.3396	0	TL	0.3396	∞
4	38	4.96	0.1562	29	4.96	0.1562
5	68	7.7879	0	47	7.7879	0
6	215	−8	0	2631	−8	0
7	3143	−8	0	622	−8	0

The results table shows that for the first three instances, SCIP was able to solve them and close the gap, except for the third instance, the gap was closed shortly after 2 h (when time limit was disabled) for the CC case, and the gap was ∞% in the LOG case. The solving time of the modified instance is mostly shorter than the original problem.

From instances with $|f^* - \overline{f^*}| > 0$, it can be implied that even if it is only one out of many constraints that was approximated, there will probably be an approximation error. These instances were approximated again with 15, 20, and 25 breakpoints, and that led to better optimal values but longer solving time. For example, when instance 1 was generated by CC with 15 points, SCIP needed 250 s to solve it and the objective value was 0.311; and with 25 points, it needed 440 s, yielding an objective value of 0.334.

An interesting finding is that the solving times of CC problems are shorter than those of LOG problems. This indicates that the models have similar performance when approximating problems with few variables using few breakpoints.

The experimental results suggest that partial PLA can produce less complicated problems with small or zero approximation error. Moreover, it was shown that LOG models have no advantage here in contrary to the case in the previous section. In the following section, similar computational experiments are performed on MIQCP problems.

3.2. Mixed Integer Variables

The PLA models, that were used in the previous section, were applied to problems where the variables involved in the quadratic terms can be integer. When introducing the breakpoints to an integer variable's interval, every integer value in the interval will be a breakpoint, so the number of breakpoints depends on the length of the interval. If this results in a large number of breakpoints, some integer values can be skipped so the number stays reasonable. Moreover, there is no need to have a convex combination of two

breakpoints since the variable cannot take noninteger values. For all noninteger variables involved in the quadratic functions, the PLA procedures are still the same.

The objective of the computational tests is the same as in the case of continuous variables: to test the uniform against the nonuniform partitioning, and to assess the performance of the partial PLA models. A summary of the test instances' statistics is given in Table 8, where they were solved by SCIP with a time limit of two hours. Instances 1, 2, and 3 were taken from the QPLIB library and the remainder are from MINLPLib instances.

Table 8. Statistics of MIQCP test instances, and the solving results by SCIP.

Instance	seq#	#v	#cons (q)	obj val	Time	Gap %
3562	1	63	42 (7)	15	TL	305
3780	2	168	72 (12)	90.6	TL	1138.3
3816	3	187	387 (24)	7.3936	TL	27.37
blend029	4	102	213 (12)	13.359	8	0
blend146	5	222	624 (24)	45.297	TL	1.87
ex1236	6	92	55 (4)	19.6	1	0
gabriel02	7	261	597 (96)	39.6	TL	22.3
sep1	8	29	31 (6)	−510.81	1	0
st_e31	9	112	135 (5)	−2	4	0
tln4	10	24	24 (4)	8.3	6	0

Based on the solving times in Table 8, it can be concluded that half of the tested instances are challenging and the other half are not. It should be mentioned that in instances 1, 2, and 10, only integer variables are involved in the quadratic functions, so no nonuniform partitioning is done for these instances (a breakpoint is introduced at every integer value in the interval). Tables 9 and 10 show the results of solving the problems produced by the convex combination and the logarithmic models with 11 breakpoints. The computations were done using CPLEX with 2 h time limit.

Table 9. Results of Solving MILP Problems Obtained from MIQCP Instances, Produced by CC and NCC Models Using 11 Breakpoints.

p#	CC			NCC ($k = 1.5$)			NCC ($k = 1.7$)		
	Time	BI	$\lvert f^* - \overline{f^*}\rvert$	Time	BI	$\lvert f^* - \overline{f^*}\rvert$	Time	BI	$\lvert f^* - \overline{f^*}\rvert$
1	TL	19.6	4.6	similar					
2	TL	F	NA	similar					
3	TL	F	NA	TL	F	NA	TL	F	NA
4	TL	12.73	0.629	TL	12.73	0.629	TL	13.359	0
5	TL	F	NA	TL	F	NA	TL	F	NA
6	360	19.6	0	68	19.6	0	237	19.6	0
7	TL	F	NA	TL	F	NA	TL	F	NA
8	109	−510.29	0.25	88	−510.08	0.73	21	−510.08	0.73
9	331	−2.019	0.019	29	−2.087	0.087	16	−2.188	0.188
10	TL	8.3	0	similar					

The results in the tables show that CPLEX failed to find an integer solution to 3 NLOG problems and 4 problems produced by the other models. For problems for which CPLEX found integer solutions, it can be observed that the problems with nonuniform partitioning are solved faster than the uniformly partitioned problems, for both logarithmic and nonlogarithmic models. The data under $\lvert f^* - \overline{f^*}\rvert$ columns show that no clear advantage can be confirmed for any of the models in the accuracy of the approximation.

Partial PLA was applied to the same set of instances and the most violated constraint of each challenging instance was approximated using CC and NLOG models. As Table 8

shows, instances 1 and 2 have a relative gap of 305% and 1138%, respectively, after two hours of solving time. After PLA, SCIP reached the time limit and resulted in gaps of 53% and 1862% for the CC problems, and 200% and 774% for the LOG problems. The rest of instances resulted in better performances by the solver without PLA.

Table 10. Results of solving MILP problems produced with LOG and NLOG models using 11 breakpoints.

p#	CC			NCC ($k = 1.5$)			NCC ($k = 1.7$)		
	Time	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$	**Time**	**BI**	$\lvert f^* - \overline{f^*}\rvert$
1	TL	18.4	3.4	similar					
2	TL	F	NA	similar					
3	TL	F	NA	TL	6.6	0.7936	TL	6.2	1.1936
4	5088	13.359	0	756	13.359	0	970	13.359	0
5	TL	F	NA	TL	F	NA	TL	F	NA
6	146	19.6	0	29	19.6	0	41	19.6	0
7	TL	F	NA	TL	F	NA	TL	F	NA
8	103	−509.72	1.09	34	−510.08	0.73	43	−510.08	0.73
9	198	−2.01	0.01	89	−2.087	0.087	103	−2.188	0.188
10	354	8.3	0	similar					

4. Discussion

To summarize the results of the computational experiments, two comparisons will be discussed. The first comparison is between PLA models using uniform against nonuniform partitioning. QCQP and MIQCQP problems were approximated using CC, NCC, LOG, and NLOG models. Most of the MILP problems that resulted from the nonuniformly partitioned model were solved to optimality faster than the problems resulting from the uniform partitioning. For example, out of 20 instances, the nonuniformly partitioned logarithmic model ($k = 1.5$) failed to find an integer solution within the time limit only three times compared to 8 failures in the uniform partitioning case. For the other 12 instances that an integer solution was found by both models, the nonuniformly partitioned model was faster 11 times. Moreover, the optimal solutions are more accurate (closer or identical to the optimal solutions of the original problems) in the nonuniform partitioning models. It is worth mentioning that the both logarithmic models were better than the convex combination models in the continuous variables problems. Despite introducing more breakpoints to LOG and NLOG (which lead to more accurate solutions), their resulting problems were solved faster.

The other comparison is to discuss whether applying PLA partially on an optimization problem can be useful ot not. We tested many QCQP and MIQCQP problems by solving them by SCIP, then we use Algorithm 1 to replace one quadratic constraints by its linear approximation, then we solve it again by SCIP. The results showed that partial PLA led to an easier problem form several instances. For these instances, the average solving time was around 4818 s without PLA, while it was 1573 and 1598 for PLA using CC and LOG models, respectively. Even though more QCQP problems were solved faster without the partial PLA, it was useful for SCIP in many QCQP problems. Improving some of the steps in Algorithm 1 can decrease the difficulty in more instances. For example the violated constraints selection rule can be improved by taking into account other aspects like current feasible solutions, variables bounds, optimal solution bounds, etc.

With the improvement of MINLP solvers, it can be concluded that full transformation of an MINLP problem into an MILP one will not be beneficial. However, doing PLA partially or using it as a supporting tool within MINLP algorithms can be useful, especially after showing that PLA itself can be improved.

5. Conclusions

The computational experiments performed on both continuous and mixed integer problems indicate promising results. It can be concluded that PLA models can be improved by using nonuniform partitioning depending on local solutions instead of uniform partitioning. Moreover, the tests have shown that some challenging QCQP and MIQCQP problems can be less challenging by applying partial PLA and replacing only one constraint by its linear approximation, which is only a small change to the original problem.

Some of the paper subjects can be considered for further research in the future. In this paper, we studied nonuniform against uniform partitioning, but it would be good research of our nonuniform partitioning was tested against other nonuniform partitioning methods. Moreover, it might be useful to do the partitioning depending on the function, so every nonlinear function could have its own domain partitioning. On the other hand, the partial PLA algorithm can be used as a tool of an existing MINLP algorithm. It can be improved by, for example, selecting a violated constraint based on all information that can be brought from the already explored nodes in the branch and bound tree.

Author Contributions: Writing—original draft, L.A.; H.M.; Writing—review & editing, L.A.; H.M. All authors have read and agreed to the published version of the manuscript.

Funding: The Deanship of Scientific Research, Qassim University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The researchers would like to thank the Deanship of Scientific Research, Qassim University for funding publication of this project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Trespalacios, F.; Grossmann, I.E. Review of mixed-integer nonlinear and generalized disjunctive programming methods. *Chem. Ing. Tech.* **2014**, *86*, 991–1012. [CrossRef]
2. Bussieck, M.R.; Vigerske, S. MINLP solver software. In *Wiley Encyclopedia of Operations Research and Management Science*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2010; pp. 1–12.
3. Smith, E.M.B.; Pantelides, C.C. Global optimisation of nonconvex MINLPs. *Comput. Chem. Eng.* **1997**, *21*, S791–S796. [CrossRef]
4. Ryoo, H.S.; Sahinidis, N.V. A branch-and-reduce approach to global optimization. *J. Glob. Optim.* **1996**, *8*, 107–138 [CrossRef]
5. Adjiman, C.S.; Androulakis, I.P.; Floudas, C.A. Global optimization of mixed-integer nonlinear problems. *AIChE J.* **2000**, *46*, 1769–1797. [CrossRef]
6. Burlacu, R.; Geißler, B.; Schewe, L. Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. *Optim. Methods Softw.* **2020**, *35*, 7–64. [CrossRef]
7. Belotti, P.; Kirches, C.; Leyffer, S.; Linderoth, J.; Luedtke, J.; Mahajan, A. Mixed-integer nonlinear optimization. *Acta Numer.* **2013**, *22*, 1–131. [CrossRef]
8. Bragalli, C.; D'Ambrosio, C.; Lee, J.; Lodi, A.; Toth, P. On the optimal design of water distribution networks: A practical MINLP approach. *Optim. Eng.* **2012**, *13*, 219–246. [CrossRef]
9. Pei, M.; Lin, P.; Du, J.; Li, X; Chen, Z. Vehicle dispatching in modular transit networks: A mixed-integer nonlinear programming model. *Transp. Res. Part Logist. Transp. Rev.* **2021**, *147*, 102240. [CrossRef]
10. Stopková, M.; Stopka, O; Klapita, V. Modeling the Distribution Network Applying the Principles of Linear Programming. 2017. Available online: https://www.semanticscholar.org/paper/Modeling-the-Distribution-Network-Applying-the-Stopkov%C3%A1-Stopka/dd47e3dd531f91c555e9ce14f732ccf232c06546 (accessed on 1 November 2021).
11. Toydas, M; Saraç, T. A mixed integer nonlinear model for air refueling optimization to save fuel in military deployment operations. *Int. J. Ind. Eng.* **2020**, *27*, 627–644.
12. Rebennack, S; Kallrath, J. Continuous piecewise linear delta-approximations for univariate functions: Computing minimal breakpoint systems. *J. Optim. Theory Appl.* **2015**, *167*, 617–643. [CrossRef]
13. Beale, E.M.L; Tomlin, J.A. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR* **1970**, *69*, 447–454.
14. Markowitz, H.M; Manne, A.S. On the solution of discrete programming problems. *Econom. J. Econom. Soc.* **1957**, *25*, 84–110. [CrossRef]

15. Dantzig, G.B. On the significance of solving linear programming problems with some integer variables. *Econom. J. Econom. Soc.* **1960**, *28*, 30–44. [CrossRef]
16. Jeroslow, R.G.; Lowe, J.K. Modelling with integer variables. *Math. Program. Oberwolfach II* **1984**, 167–184. [CrossRef]
17. Meyer, R.R. Mixed integer minimization models for piecewise-linear functions of a single variable. *Discret. Math.* **1976**, *16*, 163–171. [CrossRef]
18. Balakrishnan, A; Graves, S.C. A composite algorithm for a concave-cost network flow problem. *Networks* **1989**, *19*, 175–202. [CrossRef]
19. Misener, R.; Floudas, C.A. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Math. Program.* **2012**, *136*, 155–182. [CrossRef]
20. Castillo, P.A.C.; Castro, P.M.; Mahalec, V. Global optimization of MIQCPs with dynamic piecewise relaxations. *J. Glob. Optim.* **2018**, *71*, 691–716. [CrossRef]
21. Sundar, K.; Nagarajan, H.; Linderoth, J.; Wang, S.; Bent, R. Piecewise polyhedral formulations for a multilinear term. *Oper. Res. Lett.* **2021**, *49*, 144–149. [CrossRef]
22. Vielma, J.P.; Nemhauser, G.L. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Math. Program.* **2011**, *128*, 49–72. [CrossRef]
23. Vielma, J.P.; Ahmed, S.; Nemhauser, G. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. Res.*, **2010**, *58*, 303–315. [CrossRef]
24. Croxton, K.L.; Gendron, B.; Magnanti, T.L. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Manag. Sci.* **2003**, *49*, 1268–1273. [CrossRef]
25. Padberg, M. Approximating separable nonlinear functions via mixed zero-one programs. *Oper. Res. Lett.* **2000**, *27*, 1–5. [CrossRef]
26. Sridhar, S.; Linderoth, J.; Luedtke, J. Locally ideal formulations for piecewise linear functions with indicator variables. *Oper. Res. Lett.* **2013**, *41*, 627–632. [CrossRef]
27. Geißler, B.; Martin, A.; Morsi, A; Schewe, L. Using Piecewise Linear Functions for Solving MINLPs. *Mix. Integer. Nonlinear. Program.* **2012**, 287–314._10. [CrossRef]
28. Dahl, G.; Realfsen, B. Curve Approximation and Constrained Shortest Path Problems. Preprint (Universitetet i Oslo, Institutt for informatikk). 1996. Available online: https://www.duo.uio.no/bitstream/handle/10852/9204/1/GDahl-4.pdf (accessed on 1 November 2021).
29. Vasudeva, V. Global Optimization with Piecewise Linear Approximatio. Matser's Thesis, The University of Texas at Austin, Austin, TX, USA, 2015.
30. Nagarajan, H.; Lu, M.; Wang, S.; Bent, R; Sundar, K. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *J. Glob. Optim.* **2019**, *74*, 639–675. [CrossRef]